

Automated Verification and Validation Methods for Transmission Control Software

Madhura Medikeri and Thomas Tasky
FEV North America, Inc.

Johannes Richenhagen
FEV GmbH

ABSTRACT

With the increasing popularity of seamless gear changing and smooth driving experience along with the need for high fuel efficiency, transmission system development has rapidly increased in complexity. So too has transmission control software while quality requirements are high and time-to-market is short. As a result, extensive testing and documentation along with quick and efficient development methods are required.

FEV responds to these challenges by developing and integrating a transmission software product line with an automated verification and validation process according to the concept of Continuous Integration (CI). Hence, the following paper outlines a software architecture called "PERSIST" where complexity is reduced by a modular architecture approach. Additionally, modularity enables testability and tracking of quality defects to their root cause. To tap this potential, the software is tested, documented and built every night with a high degree of automation in order to uncover quality risks earlier in development.

The effect of that approach is shown by examples from a 7-gear series transmission project. Continuous testing and monitoring leads to a steadily increasing quality level while the project plan fulfillment can be tracked on a daily base. It is shown, that the combination of modular architecture and automated verification and validation of software helps improve the overall quality and process in the increasing complexity of the transmission system.

CITATION: Medikeri, M., Tasky, T., and Richenhagen, J., "Automated Verification and Validation Methods for Transmission Control Software," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 8(1):2015, doi:10.4271/2015-01-0163.

INTRODUCTION

Software development in the automotive industry is facing heterogeneous challenges. Demanding emission standards and fuel consumption minimization are faced with steadily more complex powertrain configurations. Control software extent and complexity are increasing at the same time. Additionally, electrification causes higher demands on functional safety functions while new comfort features intensify the nesting of so far independent functions, e.g. the usage of map data for minimized fuel consumption.

These technical requirements must be fulfilled under market conditions such as short time-to-market, country-specific conditions (e.g. fuel qualities and emission legislation) or an increasing customer need for individualized products causing more product variants.

Popular countermeasures in the automotive industry are elaborate development processes according to process models (e.g. Automotive SPICE or CMMI) and standardization of safety function development (e.g. ISO26262) or software architecture (e.g. OSEK, AUTOSAR). However, the effect of such measures is limited. About a third of all

series errors is caused by quality defects which could have been prevented by appropriate testing while troubleshooting causes up to 50% of project cost and duration (Figure 1).

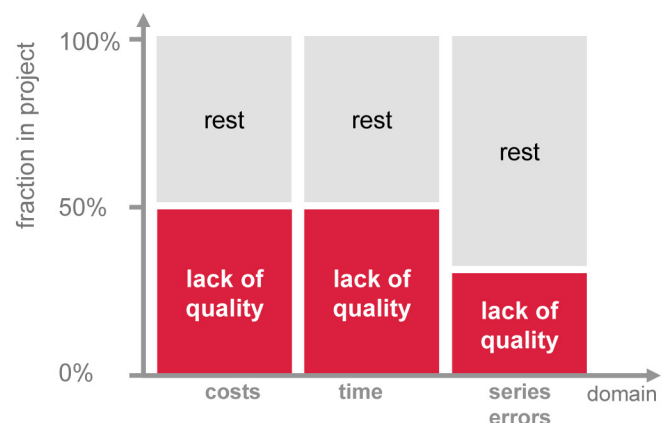


Figure 1. Effect of software quality defects on project efforts [1]

In addition, the outlined software challenges will lead to an increase of this effect. Figure 2 shows the result of a hazard and risk analysis conducted for a hybrid vehicle project at FEV GmbH.

For developing a safety concept, all hazards and risks which originate from the different powertrain components are analyzed. The hybridization with new torque components (electric motor) and a high voltage system (battery, inverter) involves a significant increase of the number of identified risks (shown in the stacked bars). Also, the severity of risks increases to a mean risk index above average (shown as dots).

As a result, new safety software functions need to be integrated which - according to the ISO 26262 standard - involves more requirements on process, functions and test activities. The outlined problems have led to a continuous increase in software development for product development from 5% in 1995 to an estimated fraction of 25% in 2025 [2]. Efficient software development on a high quality level with short development cycles is a core need for meeting market requirements of automotive products.

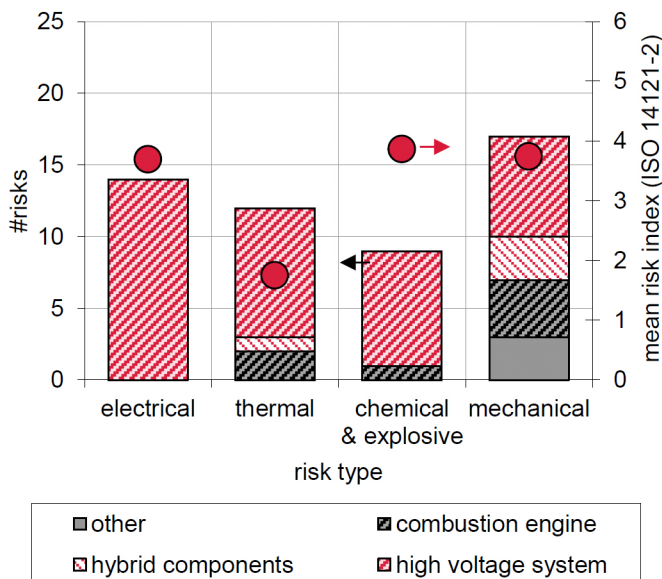


Figure 2. Safety risks caused by powertrain hybridization

REQUIREMENTS AND PROJECT CONDITIONS

As a response to the mentioned challenges, a software development approach must be developed which allows for handling high complexity levels and quality requirements at a level of agility which enables the fulfillment of customer needs.

As an example for the software extent of a series project, Table 1 lists some core facts about transmission control software for a 7-speed gearbox. Since FEV is following a product-line approach, 39 of

overall 177 software units are developed as reusable ones for future projects. An average number of 8 functional tests per unit are to be conducted which sums up to nearly 1500 test cases.

Any development activity along the V cycle including requirement specification, software architecture development, implementation, testing, integration and commissioning must be performed for that function extent.

Table 1. Data from 7-speed AMT series software development project

Number of project-specific SW units	138
Number of reusable SW units	39
Total number of test cases to be tested	1469
Total number of function frame pages	2763

The following sections outline first the chosen software development approach and focus on process details before concrete results for the given series project are shown.

APPROACH

Software Architecture and Framework for Continuous, Model-Based Integration

In response to the increasing number of software features and decreasing development cycle durations, the existing powertrain controls software architecture shall be extended to allow for flexible adaptation over all SW development steps as well as in other similar projects, thereby making it modular. This is the concept used in "PERSIST" (Powertrain control architecture Enabling Reusable Software development for Intelligent System Tailoring).

By incorporating AUTOSAR standards and safety standards such as ISO 26262 in a structure that aids in reusability over different projects, the PERSIST architecture approach helps divide the entire software into several modular elements called "compositions." These compositions consist of several "components" which are defined based on physical system attributes. The components are further divided into "functions" or SW units which are the smallest elements of the architecture.

When making these structure decisions, the software architecture team needs to address quite a few design guidelines and principles, some of which are derived from the ISO 25010 quality model. [3],[4] These guidelines ensure consistency between different projects as well as between different SW architects. [5]

Figure 3 shows an example of an application software architecture specification applied to transmission controls development software using the PERSIST framework.

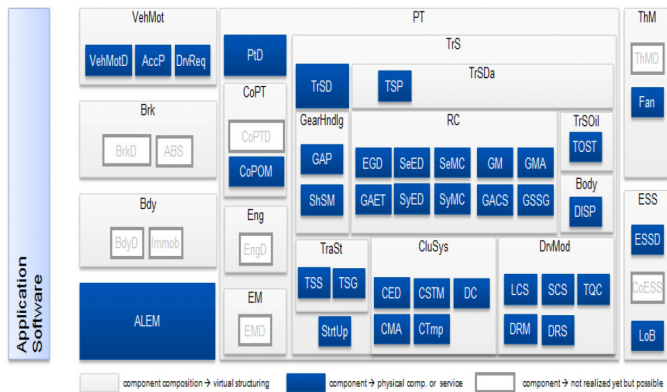


Figure 3. PERSIST APSW Architecture specification overview example

Since software development is a dynamic process, every change or feature addition needs to be integrated with the existing software and tested continuously to ensure that the quality of software is consistent. This is the motivation for “Continuous Integration” (CI). [6] CI helps avoid late detection of software errors or bugs and any associated schedule shifting. [7]

Continuous Integration when applied to a model-based environment can prove to be very beneficial. Model-based controls software has several advantages over conventional manual coding. [8] However, to mitigate any further issues that could arise due to a subjective approach of the developers, FEV has targeted to use a continuous integration approach by automating the model-based software implementation and testing methodology which has standardized the development process to a great extent. This results in reduced development and testing time as well as manual effort.

To realize this model-based design, architecture conformance tests, syntactical software verification (e.g. modeling guidelines), functional unit testing, integration tests, configuration of integration items, test planning, and the documentation and traceability of all test results must be performed at the model-level over the entire software lifecycle in an automated manner. [9]

These tasks can be well implemented using the “Nightly build” framework. Figure 4 shows how Nightly Build works.

Distributed developers commit their work into a common data repository which is version controlled. Nightly build is basically an integration server which continuously pulls updated software data from the repository and performs integration and several build tasks using a toolchain. Since it executes the SW build repeatedly every night, it is referred to as “Nightly Build”.

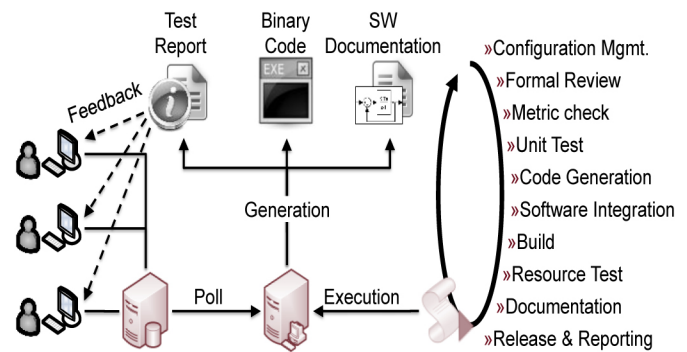


Figure 4. System Architecture of the integration framework “Nightly Build” [10]

The framework of Nightly Build is mainly based on the PERSIST development strategy for continuous software evolution (Figure 5). Short development cycles often urge software developers to react quickly to changes or updates in requirement definitions. Hence, the software development framework must be agile enough to protect for such changes within the affected release cycle. Figure 5 depicts the development approach that incorporates these needs. It is based on the daily performance of automated implementation, verification and validation and release activities for a defined SW function library. Changing requirements can be regarded with short reaction times and new release candidates can be generated which are available for validation in the field. Additionally, quality defects detected late are avoided since the verification and validation status is updated continuously.

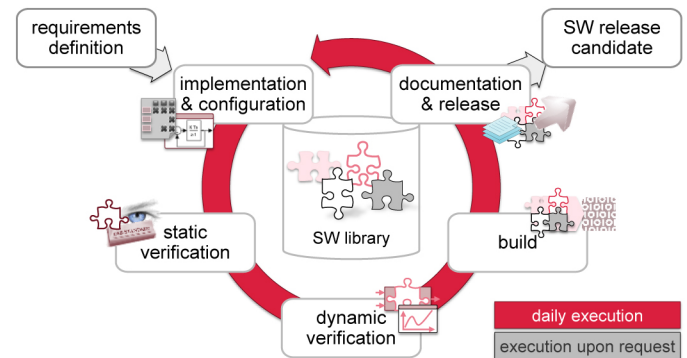


Figure 5. PERSIST development methodology for continuous software evolution

The use of this methodology along with the PERSIST architecture benefits in creating a SW library which mainly consists of all the SW units that could be reused in other similar projects. This library can continue to grow with increasing number of projects. At the same time, pre-developed SW units from this library can be used as a starting point for new relevant projects. Hence, repetitive manual effort, development time and costs are saved by building this database of reusable functions.

Nightly Build can be incorporated into most tasks of the software V cycle like configuration management, syntactical checks, static model and code analysis, software integration, unit testing and generation of software design descriptions.